Institute e-Austria in Timisoara

IeAT Report Series

A MULTI-AGENT APPROACH TO A SALES OPTIMIZATION APPLICATION

Calin SANDRU, Viorel NEGRU
Daniel POP

2003

A MULTI-AGENT APPROACH TO A SALES OPTIMIZATION APPLICATION*

Calin Sandru, Viorel Negru and Daniel Pop

Department of Computer Science, Western University of Timisoara
Research Institute e-Austria Timisoara
Bd. V. Parvan, 4, 1900 Timisoara, Romania
{sandru, vnegru, popica}@info.uvt.ro

Abstract: The paper illustrates an application of the multi agent design for a distributed application used to optimize sales people activity for a product vendor based on the customer needs and in relation with some other personal of the same company. The complex relations that may appear in the application are modeled using heterogeneous, highly specialized and autonomous entities that could be successfully designed as agents in a multi-agent environment.

Keywords: multi-agent, distributed, e-commerce, flow-control

1. INTRODUCTION

The large availability of the Internet and its related technologies made it attractive to be used to control "on-line businesses." Currently, there is a large amount of work supporting the use of agents in electronic commerce and business-to-business (B2B) domains. The large majority of the work supports automated negotiation and pricing, auctioning and transactional reasoning, and the control of supply chain management.

The application presented in the paper consists of intermediating the relations between sales people in a company and company's customers. The customers place requests for quotations that follow some life stages during their existence inside the vendor organization from inquiry to order confirmations and amendments. The job life stages are also reflected at the customer level depending on the location of the job in the vendor company. The features of such an application include:

- To enter jobs in the system
- To provide quotes (estimation or real) for the job (manually or automatically based on some price criteria)
- To move jobs based on a vendor internal flow control
- To provide good feedback to the customer
- To generate reports on various criteria
- To allow different kind of system users to view the jobs based on their needs
- To secure user access on the application
- To provide additional services like mailing and statistics
- To be linked with some legacy systems like inventory, invoicing, shipping or financial accounting.

One can identify several components to keep track of particular application tasks including job management, flow control management, reports engine, etc. Different parts of the application are written in different languages or using different technologies. Multiple people are involved at the level of design and implementation.

As illustrated above, the application has characteristics of a distributed e-commerce application where multiple components interact in order to provide the needed functionality

for various user types. There is a trend in computer science to use the multi-agent systems in order to model systems that exhibit characteristics like distribution, autonomy and interoperability using high level concepts. The agents could also be mobile, so we could benefit from having part of functionality imported and run in a particular environment (for example to do automated quotation in both customer or vendor interfaces there is a single type of agent attached to each user interface).

Researches in the e-commerce include environments for selling/buying goods (Chavez *et al.*, 1996; Maes *et al.*, 1999), general models for e-commerce using multi-agent systems (Gleizes *et al.*, 2000; Lee *et al. 1996*; Zacharia *et al.*, 2000) or particular implementations using multi-agent environments (Davis *et al.*, 2002). There also exists general purpose multi-agent systems (Kendall *et al.*, 1994; Iglesias *et al.*, 1994), or researches for methodologies to be used when implementing e-commerce applications using agents (Blake, 2001).

We found convenient to apply multi-agent results to our problem based on our previous experience illustrated in (Sandru, 1999; Sandru *et al.*, 2001). In this regard, this work is a confirmation of the usage of multi-agent systems in developing e-commerce systems.

2. AGENTS ARCHITECTURE

As presented above, there are application modules that are likely to be designed as distinct agents. This way, distinct expertise is localized at some component level and any issue that may appear is easily localized and solved. The following figure shows the various kinds of application agents.

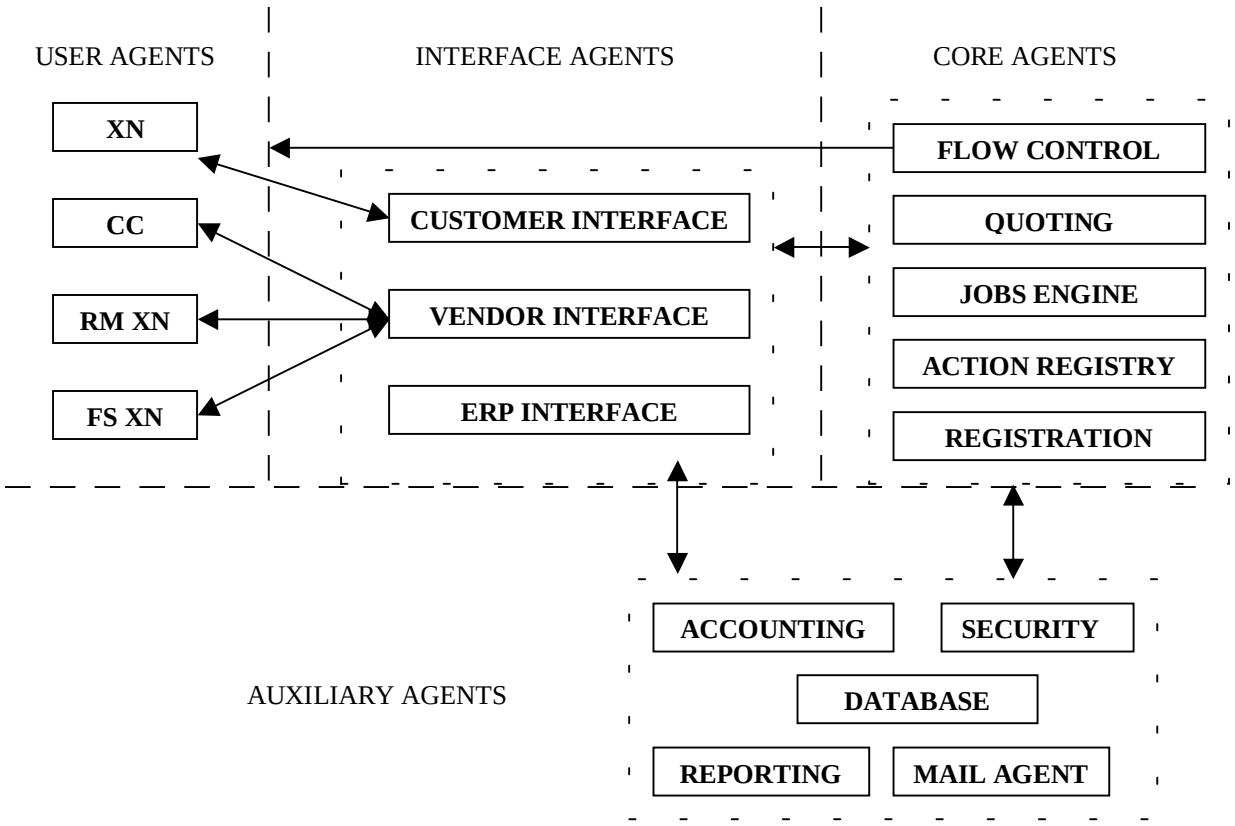

Fig. 1. Application agent based architecture

The **User Agents** guide various types of users to use the system functionality. There are mainly customer's contacts (purchasers, engineers) and vendor's users (sales people, regional managers, field sales) being able to view information related to system jobs. Any user is provided with an appropriate user interface suited for the user needs.

The customer places requests for quotation and see the job stages using the eXtraNet agent. The sales users use the CustomerConnect software in order to process the requests from the customer and to send the job to some other people for management, review, approval, etc. Other vendor users use appropriate eXtraNet versions for field sales or regional managers.

Besides allowing the user to specify its tasks, the user agents' follow a workflow controlled by a specialized agent and also provide useful information like pricing alternatives by interacting with a *Quoting* agent. This agent is a mobile one and run in the environment provided by the interface agent to generate prices based on user choices. The customer may see price estimations for the job and the sales person may quote the job for the customer using the same price criteria and variables.

The **Interface Agents** are intended to insolate the Core (business) Agents from the User Agents or to adapt the information in the format used by the application to the format used by some other legacy systems on the vendor part.

One may note that several user agents practically access the same information but filtered for that user type or third party application. Interface agents implement security mechanisms controlled by the auxiliary *Security* agent.

The **Core Agents** control the most sensitive parts of the application. The core set includes the quotation agent and the agent coping with the jobs flow as well as the agent performing all operations on jobs. The *Quotation* Agent is a mobile agent because it goes into both the XN and CC structure in order to generate prices. It is able to compute prices only in appropriate environments when a job is available. The agent manages everything pertaining to price generation, from tables to algorithms.

A special agent called *Flow Control* Agent controls the flow of jobs from requests until ordering and delivering. The rationale for this agent existence is the possibility to change the entire application way of usage by only configuring or modifying this agent. From a programmer perspective this is a big advantage because the vendor is likely to add or remove steps from the workflow. It is also helpful when adapting the application to some other domain of expertise or at some other vendor needs. In fact, the workflow agent acts based on several XML files specifying the relations between the logged user roles, the status of the interface agents and the operations an agent is allowed to execute. It also controls the chaining of operations to be executed based on agents environmental data. The next section will detail the control performed by this agent that can be seen as the "brain" of the application.

The *Action Registry* agent is a name server used for the global management of agent capabilities (actions they may perform). It stores actions descriptions in an XML based form and make the descriptions available to interface agents at initialization stage.

The *Jobs* Agent performs all operations on jobs like updating some of its components, storing it, retrieving job data, etc. This agent acts in a task based manner and in interaction with the Flow Control Agent controls which tasks have to be executed when some action occurs on a job triggered by one of the Interface Agents. Jobs Agent trigger operations in the *ERP Interface* Agent in order to make data consistent with the internal ERP application.

The *Registration* agent is only responsible with registering vendor users and customers' contacts, managing their roles and access rights.

Auxiliary Agents performs some useful tasks for the application and they are intended only to act as clients. Other agents requests operations from these agents and they act in consequence based on their internal rules.

3. CONTROLLING THE WORKFLOW ON THE USER AGENTS

We pointed out that one main benefit of this multi-agent architecture is the opportunity to specify the user agents' workflow by using a specialized agent: the *Flow Control* Agent. This section will detail its relationship with the user agents.

A user agent has the following structure:

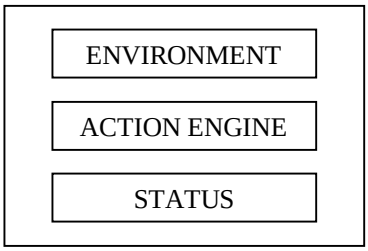

Fig. 2. User Agent

The environment is the agent data context. It may be modified as a result of agent internal actions or as a result of calling external agent actions.

The action engine executes actions either triggered by interface buttons or *pre/postconditions* of those actions. *Preconditions* and *postconditions* are list of actions to be executed before or after the action body. Preconditions return an error code. When "*ok*", the action body is executed. Otherwise the action is cancelled. Postconditions test the error code of the action and perform an appropriate set of actions for that error code.

One action is related to an agent implementing it. The action engine knows where to pass the action for execution along with its argument. The action arguments are specified in the action definition along with other optional attributes (for example if the action must be started in a new window of specified dimensions). Before calling an action, the arguments are collected from the agent environment or from other local environment considered as having higher priority. For example, actions in a postcondition will receive primarily arguments from a context returned by the main action execution before the action engine looks for arguments in the global environment.

Actions effects are either interface updates or changes in the environment. The action definition specifies what to update in the environment after a successful execution.

Below is the XML description of two related sample actions:

```
<action name="start_new_job">
   <agent>VendorInterface</agent>
   <type>window</type>
   <window>
      <width>500</width>
      <height>300</height>
      <title>New Job</title>
   </window>
   <arg>session_id</arg>
   <arg>user/user_id</arg>
   <precondition>ask_save_if_dirty</precondition>
   <postcondition>post_start_job</postcondition>
```

```
</action>

<action name="new_job">
   <agent>JobsEngine</agent>
   <arg>session_id</arg>
   <arg>user/user_id</arg>
   <arg>company_id</arg>
   <arg>job_category</arg>
   <arg type="optional">job_type</arg>
   <update parent="*" local="job"/>
   <postcondition>post_new_job</postcondition>
   <postcondition>post_display_job</postcondition>
</action>
```

and the <post_start_job> postcondition:

```
<postcondition name="post_start_job">
   <if local="error_code" value="ok">
      <action>new_job</action>
   </if>
</postcondition>
```

The descriptions above forces the following chain of operations to happen in the CustomerConnect user agent:

- The user press an agent interface button called “New Job”
- The action engine begins to execute the *start_new_job* action. It first executes the precondition *ask_save_if_dirty* meaning that the user is asked to save the current loaded job before starting a new one.
- The action engine opens a new window for the action and retrieves the argument values from the specified paths in the environment (*session_id* and *user/user_id*).
- The action engine passes the action and its arguments to the *VendorInterface* agent to execute it.
- Executing the action means selecting some values: *company_id* and *job_category* to be returned to the caller along with an *error_code*. When returned, these values go into a local action context. This context is passed to the action postconditions and has more priority when searching from arguments in the postcondition’s actions.
- The *post_start_job* postcondition is going to be executed. As specified in the postcondition body above, if the received *error_code* is “*ok*” then the *new_job* action is started. The same cycle begins, but the *new_job* action also updates the global environment of the agent for the key *job* with an empty job as received from the JobsEngine agent.
- Eventually, *new_job* postconditions are then executed allowing the agent to display the job on the user interface.

The Flow Control agent may overwrite the original action description stored by the Action Registry. This means that the same action has different preconditions and postconditions based on the interface agent status and the access rights of that agent. Thus, starting a new job from the CustomerConnect used by a salesman may differ from the same operation performed by a manager by simply removing some preconditions.

Usually an action changes the agent’s internal status. The Flow Control agent specifies which actions should be callable by an interface agent in a particular status by listing, in an XML entry, the actions for that agent status. This is a mechanism through which the agent may update its interface by enabling or disabling functionality according to the options received.

The actions tasks are restricted to follow one or more *constraints*. The constraints allows for example to display a job or some part of the jobs in a read only form. The format of the constraints is dependant of the action operations. Different actions may have completely different XML constraints description.

The above exemplified agent interactions allow user agents exhibiting behaviors they were not initially designed to have, by only changing the interface agents and updating the tables of the flow control agent. For example, there is no need to change the user agent when starting a new job as a revision of an existing job. The *start_new_job* action performed by the *VendorInterface* agent may be upgraded to select the job type and the XML-ed description of the new action new_revision is detailed below:

```
<action name="start_revision">
<agent>VendorInterface</agent>    <type>window</type>
   <window>
      <width>400</width>
      <height>150</height>
      <title>New Revision</title>
   </window>
   <arg>session_id</arg>
   <arg>user/user_id</arg>
   <arg value="Revision">job_type</arg>
   <arg name="base_job_id">job/job_id</arg>
   <arg>job/job_no</arg>
   <precondition>ask_save_if_dirty</precondition>
   <postcondition>post_revision</postcondition>
</action>

<postcondition name="post_revision">
   <if local="error_code" value="ok">
      <action>new_revision</action>
   </if>
<postcondition>

<action name="new_revision">
   <agent>JobsEngine</agent>
   <arg>session_id</arg>
   <arg>user/user_id</arg>
   <arg type="optional">base_job_id</arg>
   <arg type="optional">job_type</arg>
   <arg type="optional">job_no</arg>
   <update parent="*/job" local="job_id"/>
   <update parent="*/job" local="job_no"/>
   <update parent="*/job" local="date_created"/>
   <postcondition>post_new_job</postcondition>
   <postcondition>post_display_job</postcondition>
</action>
```

The *start_revision* action retrieves from the environments some values it needs in order to build relation between the old job and the revision. These are then passed to the *new_revision* action in order to prepare the new context for the revision version. The updates in the agent environment specify that some of the environmental data must be changed to the new values: *job_id* becomes the id of the revision job, *job_no* becomes a modified value of the one provided by the *start_revision* action and the *date_created* is reset. At this point the user agent is allowed to do similar operations as after the *new_job* case.

4. CONCLUSIONS

The advantages of the multi-agent architecture allowed us to build a system exhibiting characteristics as follows:

- Highly modular with autonomous entities.
- Open environment in the sense that new agents may be added to provide new functionality.
- Easy to manage and implement because any agent may be written in it's own programming language by different people.
- Flexibility in configuration of user agents. New operation at the user level could be easily added with minimum user impact.

ACKNOWLEDGEMENTS

This work was partially supported by: the Romanian Government's INFOSOC grant no. 61/2002, by the Austrian Ministries BMBWK project no. GZ 45.527/1-VI/B/7a/02 and BMWA project GZ no. 98.244/1-I/18/02. Our industry partner was represented by WebQuote.com SRL Timisoara.

REFERENCES

1. **Blake, M. Brian.** (2001) Innovations in Software Agent-Based B2B Technologies, Workshop On Agent-Based Approaches to B2B at The 5th International Conference on *Autonomous Agents* May 29, 2001
2. **Chavez A. and P. Maes**. (1996) Kasbah: An agent marketplace for buying and selling goods. *In The First International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology, PAAM'96* April 1996
3. **Davis, D.N., Y. Luo, K. Liu.** (2002) Combining KADS with ZEUS to Develop a Multi-Agent E-Commerce Application. In Journal of E-Commerce Research, 2002
4. **Kendall, E., M. Malkoun, and C.H. Jiang**. (1994) A methodology for developing agent-based systems. *In J.W. Perram and J.P Muler, editors, Distributed Software. Agents and Applications, 6th European Workshop on MAAMAW*, pages 85-99, 1994
5. **Gleizes, M.P., J.L.Pezet and P. Glize**. (2000) An adaptive multi-agent tool for electronic commerce. *In Proceedings of the IEEE 9th International Workshop on Enabling Technologies: Infrastructure for collaborative Enterprises (WET ICE'00)*, 2000
6. **Glushko, R., J. M. Tenenbaum and B. Meltzer**. (1999) An XML framework for agent based e-commerce. Communication of the ACM, 42(3):106-114, March 1999
7. **Iglesias, C., J. Gonzales and J. Velasco**. (1994) Mix: A general purpose multi-agent architecture. *In LNCS 1069, Distributed Software Agents and Applications, 6th European Workshop on MAAMAW*, 1994
8. **Lee, J.G., J.Y. Kang and E.S. Lee**. (1997) ICOMA: An open infrastructure for agent-based intelligent electronic commerce on the Internet. *In Proceedings of the International Conference on Parallel and Distributed Systems (ICPADS'97)*, 1997.
9. **Maes, P., R. Guttman and A. Moukas**. (1999) Agents that buy and sell: Transforming commerce as we know it. *Communication of the ACM*, pages 81-87, March 1999.
10. **Sandru, C**. (1999) A multi-agent oriented approach in mathematical problem solving. *In Proceedings of the 21st International Conference on Information Technology Interfaces*, Croatia, 1999, pages 107-112.
11. **Zacharia, G., A, Moukas, P. Boufounos and P. Maes**. (2000) Dynamic pricing in a reputation brokered agent mediated knowledge marketplace. *In HICSS-33*, 2000.
12. **Sandru, C., V. Negru, D. Pop**. (2001) Intermediating commerce transactions using multi-agent techniques, *Proceedings of the 2nd International Workshop of Central and Eastern Europe on Multi-Agent Systems, CEEMAS 01*, September 26 - 29, 2001, Krakow, Poland, published in proceedings, eds. Barbara Dunin-Keplitz, Edward Nawarecki, Leyko Pbs., 2001, ISBN 83-915953-0-7, pp 377-385;